

Making your own tool using **SwiftSyntax**

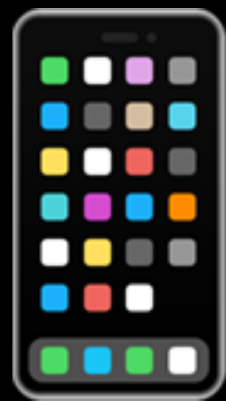
The background of the slide is split vertically. The left half shows a person with dark hair, wearing a dark jacket, smiling slightly. The right half is dark with a large, semi-transparent red hexagon and a blue circle. The word 'mercari' is faintly visible in the bottom right corner of the dark area.

Hi, I'm Yusuke

@kitasuke

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

App development



Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Source code



Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Tools

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Tools help make programming easier

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

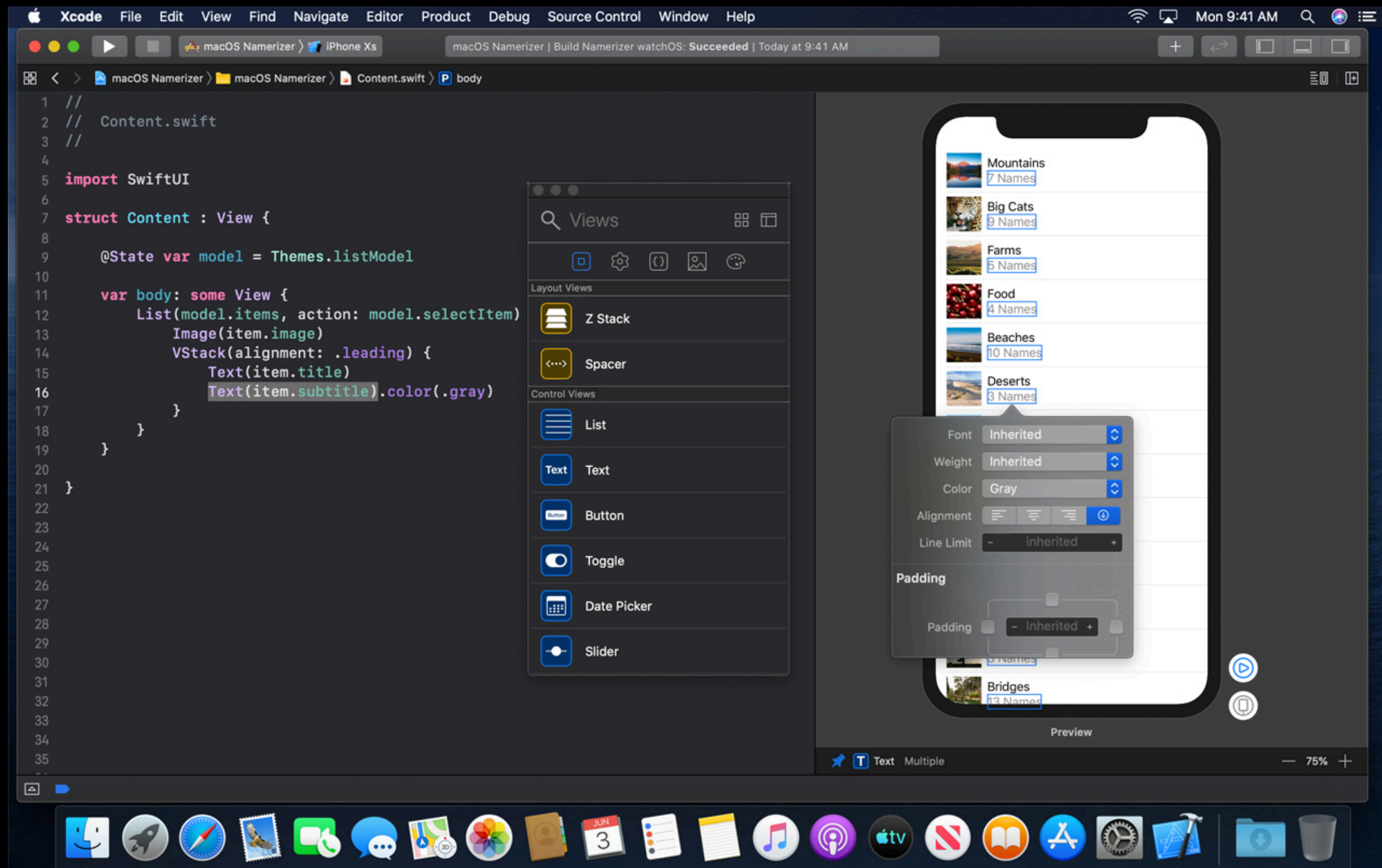
Tools using **SwiftSyntax**

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SwiftUI Previews

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SwiftUI Previews



Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

swift-format

SE-0250: Swift Code Style Guidelines and Formatter

swift-format



```
let number: Int = 5
```



```
let number: Int = 5  
let number : Int = 5
```

SwiftConst

Find in repeated strings that could be replaced by a constant

SwiftConst

```
$ swiftconst run  
other occurrence(s) of "error" found in: main.swift:7:11  
other occurrence(s) of "help" found in: main.swift:18:19  
other occurrence(s) of "error" found in: main.swift:19:28  
other occurrence(s) of "help" found in: main.swift:21:19
```

SwiftSyntax

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

```
tilViewController = segue!.destinationViewContro
```

```
l as String  
l as String
```

**SourceKitService
Terminated**

**Editor functionality
temporarily limited.**

You selected cell **#0!**

What's **SwiftSyntax** for?

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Code modifier

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Code analyzer

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

What's **SwiftSyntax**?

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SwiftSyntax is a set of Swift bindings for the libSyntax library. It allows for Swift tools to parse, inspect, generate, and transform Swift source code.

libSyntax

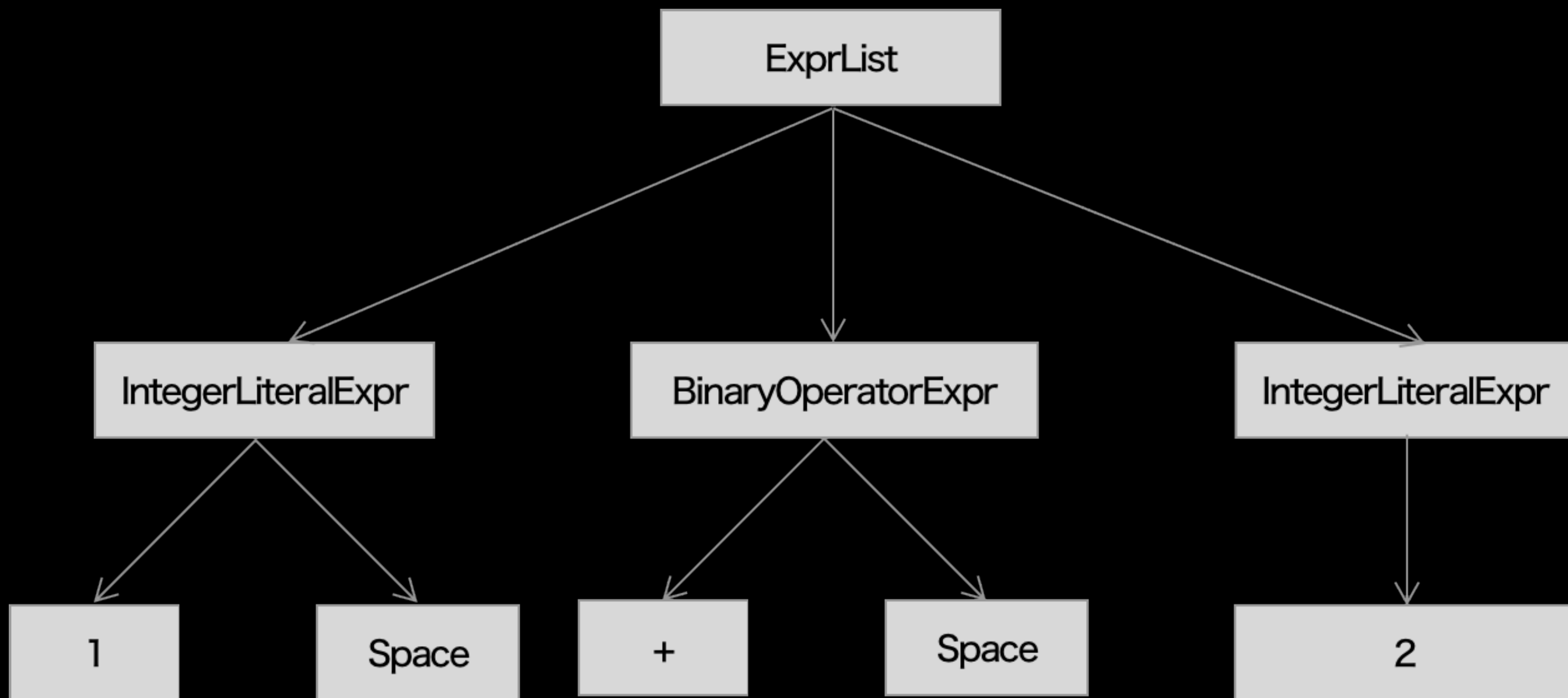
Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

***libSyntax library aims to
represent the syntax tree of the
source file***

Syntax Tree

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

1 + 2



SwiftSyntax parses **Swift** source code to syntax tree

How to emit syntax tree

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

```
$ swift -frontend emit-syntax
```

Parse input file(s) and emit the Syntax tree(s) as
JSON

```
$ swift -frontend emit-syntax
```

```
// 1 + 2
- SourceFile
  - CodeBlockItemList
    - CodeBlockItem
      - SequenceExpr
        - ExprList
          - IntegerLiteralExpr
            - integer_literal: 1
            - trailingTrivia: Space
          - BinaryOperatorExpr
            - oper_binary_spaced: +
            - trailingTrivia: Space
          - IntegerLiteralExpr
            - integer_literal: 2
```

Swift AST Explorer

Swift AST to HTML conversion

Swift AST Explorer

The screenshot displays the Swift AST Explorer interface. On the left, a code editor shows Swift source code for a `BlackjackCard` struct. The code includes imports, nested enumerations for `Suit` and `Rank`, a nested struct `Values`, and a `description` property. The `Suit` enumeration is highlighted in blue. On the right, the 'Syntax' tab shows the AST structure. The tree starts with `CodeBlockItemList`, followed by `CodeBlockItem`, `ImportDecl`, `AccessPathComponent`, and `Foundation`. The `BlackjackCard` struct is represented by a `StructDecl` node. Its `MemberDeclBlock` contains a `DeclList` with an `enum` declaration for `Suit`. A tooltip for the `enum` node shows its metadata: `kind: enumKeyword`, `leadingTrivia:`, `trailingTrivia:`, and `text: enum`. The `Suit` enumeration is further detailed with a `TypeInheritanceClause`, an `InheritedTypeList` containing `Character`, and a `DeclList` for its members.

```
1 import Foundation
2
3 struct BlackjackCard {
4     // nested Suit enumeration
5     enum Suit: Character {
6         case spades = "♠", hearts = "♥", diamonds = "♦", clubs = "♣"
7     }
8
9     // nested Rank enumeration
10    enum Rank: Int {
11        case two = 2, three, four, five, six, seven, eight, nine, ten
12        case jack, queen, king, ace
13
14        struct Values {
15            let first: Int, second: Int?
16        }
17
18        var values: Values {
19            switch self {
20            case .ace:
21                return Values(first: 1, second: 11)
22            case .jack, .queen, .king:
23                return Values(first: 10, second: nil)
24            default:
25                return Values(first: self.rawValue, second: nil)
26            }
27        }
28    }
29
30    // BlackjackCard properties and methods
31    let rank: Rank, suit: Suit
32    var description: String {
33        var output = "suit is \(suit.rawValue),"
34        output += " value is \(rank.values.first)"
35        if let second = rank.values.second {
36            output += " or \(second)"
37        }
38        return output
39    }
40 }
41
```

Swift version 4.1-dev (LLVM c4ec2ab808, Clang af436f313e, Swift be19556ce8)
Target: x86_64-unknown-linux-gnu

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Recap

Recap

→ SwiftSyntax is a parsing tool

Recap

- SwiftSyntax is a parsing tool
 - Emits syntax tree

Recap

- SwiftSyntax is a parsing tool
 - Emits syntax tree
- Allows to inspect and transform source code

How **SwiftSyntax** works?

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Usage via *SwiftPM*

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Package.swift

```
import PackageDescription

let package = Package(
    name: "MyTool",
    dependencies: [
        .package(url: "https://github.com/apple/swift-syntax.git",
            .exact("<#Specify Release tag#>")),
    ],
    targets: [
        .target(name: "MyTool", dependencies: ["SwiftSyntax"]),
    ]
)
```

Interfaces

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SyntaxParser

Parsing Swift source

SyntaxParser

```
/// Parses the file `URL` into a full-fidelity Syntax tree.
///
/// - Parameters:
///   - url: The file URL to parse.
///   - diagnosticEngine: Optional diagnostic engine to where the parser will
///     emit diagnostics
/// - Returns: A top-level Syntax node representing the contents of the tree,
///   if the parse was successful.
/// - Throws: `ParserError`
public static func parse(_ url: URL,
    diagnosticEngine: DiagnosticEngine? = nil) throws -> SourceFileSyntax
```


SourceFileSyntax

Structure of source file

SourceFileSyntax

```
// 1 + 2 in sample.swift
```

```
- SourceFileSyntax
  - CodeBlockItemListSyntax
    - CodeBlockItemSyntax
      - SequenceExprSyntax
        - ExprListSyntax
          - IntegerLiteralExprSyntax
            - integerLiteral: 1
            - trailingTrivia: Space
          - BinaryOperatorExprSyntax
            - spacedBinaryOperator: +
            - trailingTrivia: Space
          - IntegerLiteralExprSyntax
            - integerLiteral: 2
```

Syntax

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

IntegerLiteralExprSyntax

```
public struct IntegerLiteralExprSyntax:
    ExprSyntax, _SyntaxBase, Hashable {
    enum Cursor: Int {
        case digits
    }

    public var digits: TokenSyntax {...}
    ...
}
```

TokenSyntax

```
/// A Syntax node representing a single token.
public struct TokenSyntax: _SyntaxBase, Hashable {
    /// The text of the token as written in the source code.
    public var text: String {
        return tokenKind.text
    }
    ...
}
```

Syntaxes

IdentifierExprSyntax, ArrayExprSyntax,
DictionaryExprSyntax, IntegerLiteralExprSyntax,
BooleanLiteralExprSyntax, FunctionCallExprSyntax,
StringLiteralExprSyntax, ClassDeclSyntax, StructDeclSyntax,
ProtocolDeclSyntax, VariableDeclSyntax, EnumDeclSyntax,
SwitchStmtSyntax, IfStmtSyntax ...

SyntaxVisitor

Visiting each syntax

SyntaxVisitor

```
/// Visiting `IntegerLiteralExprSyntax` specifically.  
/// - Parameter node: the node we are visiting.  
/// - Returns: how should we continue visiting.  
mutating func visit(_ node: IntegerLiteralExprSyntax) -> SyntaxVisitorContinueKind {  
    return .visitChildren  
}  
  
/// The function called after visiting `IntegerLiteralExprSyntax` and its descendents.  
/// - node: the node we just finished visiting.  
mutating func visitPost(_ node: IntegerLiteralExprSyntax) {}
```


SyntaxRewriter

Rewriting each syntax

SyntaxRewriter

```
open func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {  
    return visitChildren(node) as! ExprSyntax  
}
```

SyntaxFactory

Make each syntax

SyntaxFactory

```
public static func makeIntegerLiteralExpr(digits: TokenSyntax)
-> IntegerLiteralExprSyntax {
    let layout: [RawSyntax?] = [
        digits.raw,
    ]
    let raw = RawSyntax.createAndCalcLength(
        kind: .integerLiteralExpr,
        layout: layout,
        presence: .present
    )
    let data = SyntaxData.forRoot(raw)
    return IntegerLiteralExprSyntax(data)
}
```

Usecases

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Odd number inspector 🚦

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Odd number inspector

```
// Use even instead, not odd  
view.frame.height = 43
```

OddNumberInspector

```
struct OddNumberInspector: SyntaxVisitor {
    mutating func visitPost(_ node: IntegerLiteralExprSyntax) {
        // do nothing if it's even
        guard let integer = Int(node.digits.text),
              integer % 2 == 1 else {
            return
        }
        print("Use even instead, not odd")
    }
}
```


OddNumberInspector

```
struct OddNumberInspector: SyntaxVisitor {  
    mutating func visitPost(_ node: IntegerLiteralExprSyntax) {  
        // do nothing if it's even  
        guard let integer = Int(node.digits.text),  
              integer % 2 == 1 else {  
            return  
        }  
        print("Use even instead, not odd")  
    }  
}
```

OddNumberInspector

```
struct OddNumberInspector: SyntaxVisitor {  
    mutating func visitPost(_ node: IntegerLiteralExprSyntax) {  
        // do nothing if it's even  
        guard let integer = Int(node.digits.text),  
              integer % 2 == 1 else {  
            return  
        }  
        print("Use even instead, not odd")  
    }  
}
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// walk SourceFileSyntax with visitor
var visitor = OddNumberInspector()
sourceFile.walk(&visitor)
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// walk SourceFileSyntax with visitor
var visitor = OddNumberInspector()
sourceFile.walk(&visitor)
```

Odd number inspector

```
// Use even instead, not odd  
view.frame.height = 43
```

Ten multiplication formatter 🦾

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Ten multiplication formatter

```
let i: Int = 1 // -> 10
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {  
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {  
        guard let integer = Int(node.digits.text) else {  
            return node  
        }  
  
        let digits = SyntaxFactory.makeIntegerLiteral(  
            String(integer * 10),  
            leadingTrivia: node.leadingTrivia ?? .zero,  
            trailingTrivia: node.trailingTrivia ?? .zero  
        )  
        return IntegerLiteralExprSyntax {  
            $0.useDigits(digits)  
        }  
    }  
}
```


TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {
  override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {
    guard let integer = Int(node.digits.text) else {
      return node
    }

    let digits = SyntaxFactory.makeIntegerLiteral(
      String(integer * 10),
      leadingTrivia: node.leadingTrivia ?? .zero,
      trailingTrivia: node.trailingTrivia ?? .zero
    )
    return IntegerLiteralExprSyntax {
      $0.useDigits(digits)
    }
  }
}
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {  
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {  
        guard let integer = Int(node.digits.text) else {  
            return node  
        }  
  
        let digits = SyntaxFactory.makeIntegerLiteral(  
            String(integer * 10),  
            leadingTrivia: node.leadingTrivia ?? .zero,  
            trailingTrivia: node.trailingTrivia ?? .zero  
        )  
        return IntegerLiteralExprSyntax {  
            $0.useDigits(digits)  
        }  
    }  
}
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {  
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {  
        guard let integer = Int(node.digits.text) else {  
            return node  
        }  
  
        let digits = SyntaxFactory.makeIntegerLiteral(  
            String(integer * 10),  
            leadingTrivia: node.leadingTrivia ?? .zero,  
            trailingTrivia: node.trailingTrivia ?? .zero  
        )  
        return IntegerLiteralExprSyntax {  
            $0.useDigits(digits)  
        }  
    }  
}
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// visit SourceFileSyntax
var visitor = TenMultiplicationFormatter()
let modifiedSourceFile = TenMultiplicationFormatter().visit(sourceFile)
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// visit SourceFileSyntax
var visitor = TenMultiplicationFormatter()
let modifiedSourceFile = TenMultiplicationFormatter().visit(sourceFile)
```

Ten multiplication formatter

```
let i: Int = 10 // <- 1
```

**Tools make your code
much better 😎**

**Tools sometimes are
better corder than you 😏**

Improvements

- Speeding up SwiftSyntax by using the parser directly
- Integrating libSyntax into the compiler pipeline
- Declarative syntax creation using function builders

Summary

Summary

→ Take benefit of statically typed language

Summary

- Take benefit of statically typed language
 - Useful tools make better apps

Summary

- Take benefit of statically typed language
 - Useful tools make better apps
- SwiftSyntax's improving significantly

References

- <https://github.com/apple/swift-syntax>
- <https://www.slideshare.net/kitasuke/integrating-libsyntax-into-the-compiler-pipeline>

Thank you!